Storypoint Problem Exploration - mule

September 5, 2024

1 Storypoint Prediction: Problem Exploration

1.1 Problem Statement

In modern agile development settings, software is developed through repeated cycles (iterative) and in smaller parts at a time (incremental), allowing for adaptation to changing requirements at any point during a project's life. A project has a number of iterations (e.g. sprints in Scrum). Each iteration requires the completion of a number of user stories, which are a common way for agile teams to express user requirements.

There is thus a need to focus on estimating the effort of completing a single user story at a time rather than the entire project. In fact, it has now become a common practice for agile teams to go through each user story and estimate its "size". Story points are commonly used as a unit of measure for specifying the overall size of a user story.

1.2 Problem Formulation

Input: A string of length N that contains a story's name and description $C = \{c_1, c_2, c_3, ..., c_n\}$. For each story, a set of text embeddings that contains features $E = \{e_1, e_2, e_3, ..., e_m\}$ extracted from C has been provided.

Output: A natural number P associated with the story point of that user story

1.3 Dataset Information

Text Embeddings: Text embeddings are a way to convert words or phrases from text into a list of numbers, where each number captures a part of the text's meaning. The dataset has been preprocessed and converted into two kinds of text embeddings. You can choose to work with either of them or both: - Doc2Vec: Input strings are transformed into fixed-length vectors of size 128. These vectors capture the semantic meaning of words and their relationships within a document. - Look-upTable: Input strings are transformed into fixed-length vectors of size 2264. These vectors are obtained via transforming each word in the input strings into an identifier number, then padded to the length of the longest sample.

Dataset Structure & Format: Storypoint Estimation Dataset is stored in 3 folders labeled *raw data*, *look-up*, and *doc2vec*. Within each folder are 3 CSV files for training, testing, validation. Each csv file has the following columns: - `issuekey` : The unique identifier for a story. - `storypoint`: The correct number of storypoint. - An embedding column (`embedding` or `doc2vec`) contains text embedding vectors. The raw data csv will not have this and instead contain two columns with story `name` and `description`.

1.4 Exploration

1.4.1 Raw data exploration

[ ]:
```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns

from sklearn.feature_extraction.text import CountVectorizer
```

Output exploration

[ ]:
```
# Import raw data from the CSV file

project_name = 'mule'

all_data = pd.concat([pd.read_csv('data/' + project_name + '/' + project_name +␣
 ↪'_train.csv'),
                      pd.read_csv('data/' + project_name + '/' + project_name +␣
 ↪'_valid.csv'),
                      pd.read_csv('data/' + project_name + '/' + project_name +␣
 ↪'_test.csv')])

print('Check the shape of the dataset', all_data.shape)
```

```
Check the shape of the dataset (889, 4)
```

[ ]:
```
all_data.drop(['issuekey'], axis=1, inplace=True)
all_data.head()
```

[ ]:
```
                                              title  \
0   implement true multicast functionality processor
1  mule support transaction resources used spring…
2   include validation module mule core distribution
3  untilsuccessful set exception payload last exc…
4      untilsuccessful support synchronous use cases

                                         description  storypoint
0  currently processes messages sequentially coul…           8
1  know common use case bridge pattern read messa…           8
2  emiliano created validation module something t…           8
3  title says see thread discussions deprecate dl…           5
4  imagine simple synchronous http proxy use case…           8
```

First, let take a look at the distribution of the story point:

Interpretation of Skewness Values:

- **Skewness > 0**: Right-skewed distribution.
- **Skewness < 0**: Left-skewed distribution.

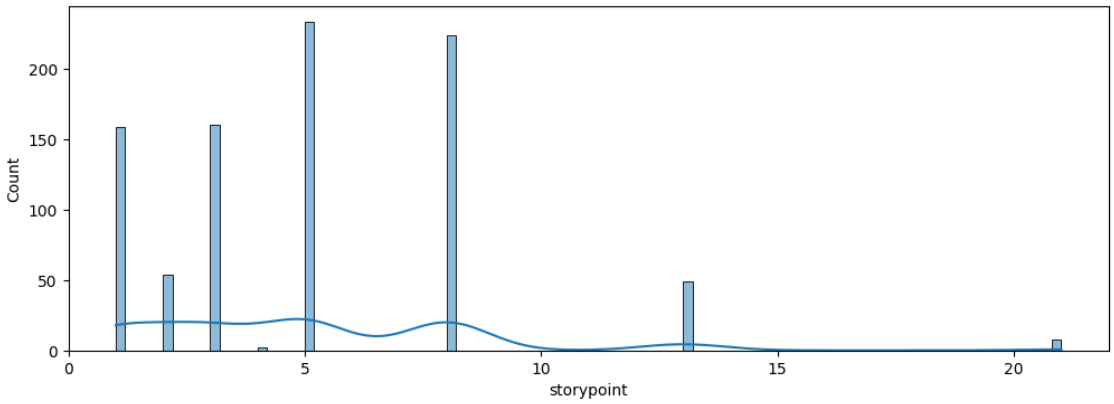
- **Skewness = 0**: Symmetrical distribution (like a normal distribution).

Interpretaion of kurtosis: - **Leptokurtic (Kurtosis > 3)**: The distribution has heavier tails and a sharper peak than the normal distribution. Data points are more likely to produce extreme values. The distribution has a higher peak and fatter tails. - **Platykurtic (Kurtosis < 3)**: The distribution has lighter tails and a flatter peak than the normal distribution. Data are fewer extreme values compared to a normal distribution. - **Mesokurtic (Kurtosis 3)**: The distribution has a similar kurtosis to the normal distribution, indicating a moderate level of outliers.

[ ]:
```
# Draw a histogram of the story points
plt.figure(figsize=(12, 4))
plt.xticks(np.arange(0, max(all_data['storypoint']) + 1, 5))
sns.histplot(all_data['storypoint'], bins=100, kde=True)

print('Skewness:', all_data['storypoint'].skew())
print('Kurtosis:', all_data['storypoint'].kurt())
```

```
Skewness: 1.2748024522567163
Kurtosis: 2.855540938407874
```



[ ]:
```
tmp = pd.concat([all_data['storypoint'].value_counts(),
                 all_data['storypoint'].value_counts() / all_data.shape[0] *␣
 ↪100],
                axis=1, keys=['Counts', 'Percentage (%)'])
tmp.head(20)
```

[ ]:

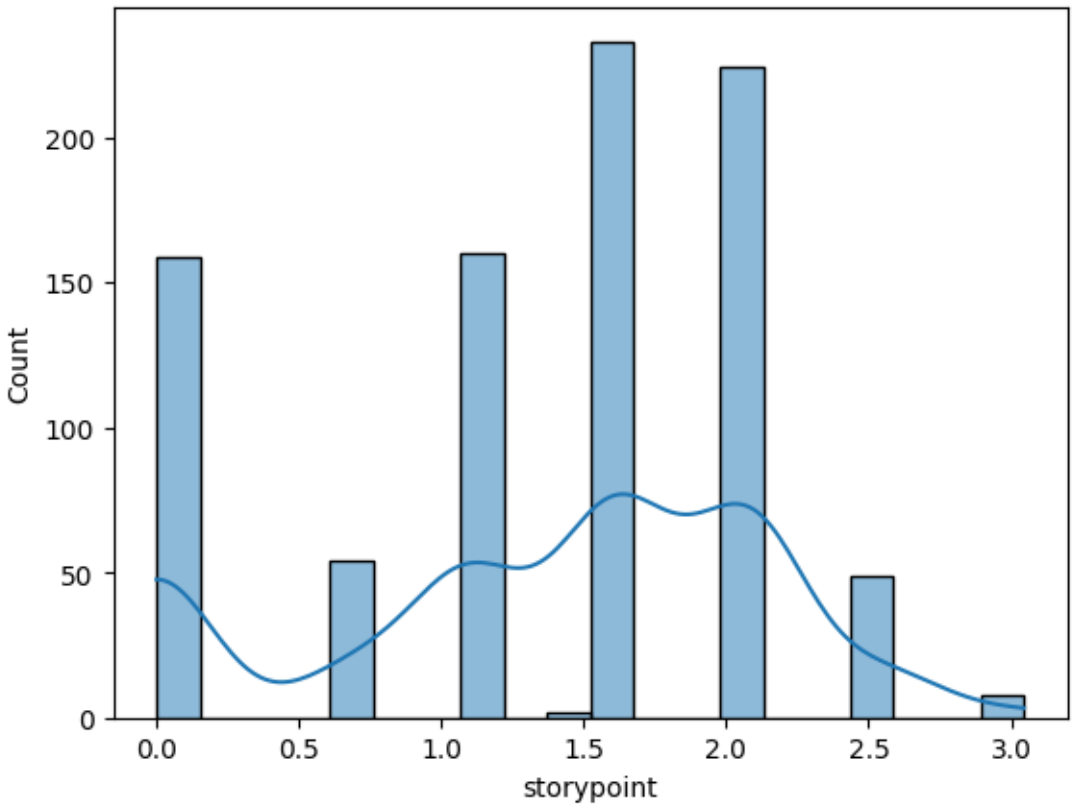
storypoint	Counts	Percentage (%)
5	233	26.209224
8	224	25.196850
3	160	17.997750
1	159	17.885264
2	54	6.074241
13	49	5.511811

```
21              8         0.899888
4               2         0.224972
```

Its distribution of the label is a little bit **right-skewed** and **playtykurtis**. I will try to use log-scale, maybe it's better

[ ]:
```
sns.histplot(np.log(all_data['storypoint']), bins=20, kde=True)
```

[ ]:
```
<Axes: xlabel='storypoint', ylabel='Count'>
```



[ ]:
```
print('Skewness:', np.log(all_data['storypoint']).skew())
print('Kurtosis:', np.log(all_data['storypoint']).kurt())
```

```
Skewness: -0.46738335118599855
Kurtosis: -0.7267872020080017
```

Nah, the kurtosis now becomes too **leptokurtis**

Input exploration The input of this problem is 2 texts: title and description. First we will find some statistics:

[ ]:
```
title_lengths = all_data['title'].apply(lambda x: len(x.split(' ')))
print('Title analysis:')
print('   - Mean length:', round(title_lengths.mean()))
print('   - Min length:', title_lengths.min())
print('   - Max length:', title_lengths.max())

description_lengths = all_data['description'].apply(lambda x: len(x.split(' '))␣
 ↪if type(x) != float else 0)
print('Description analysis:')
print('   - Mean length:', round(description_lengths.mean()))
print('   - Min length:', description_lengths.min())
print('   - Max length:', description_lengths.max())
```

```
Title analysis:
   - Mean length: 6
   - Min length: 1
   - Max length: 17
Description analysis:
   - Mean length: 31
   - Min length: 0
   - Max length: 833
```

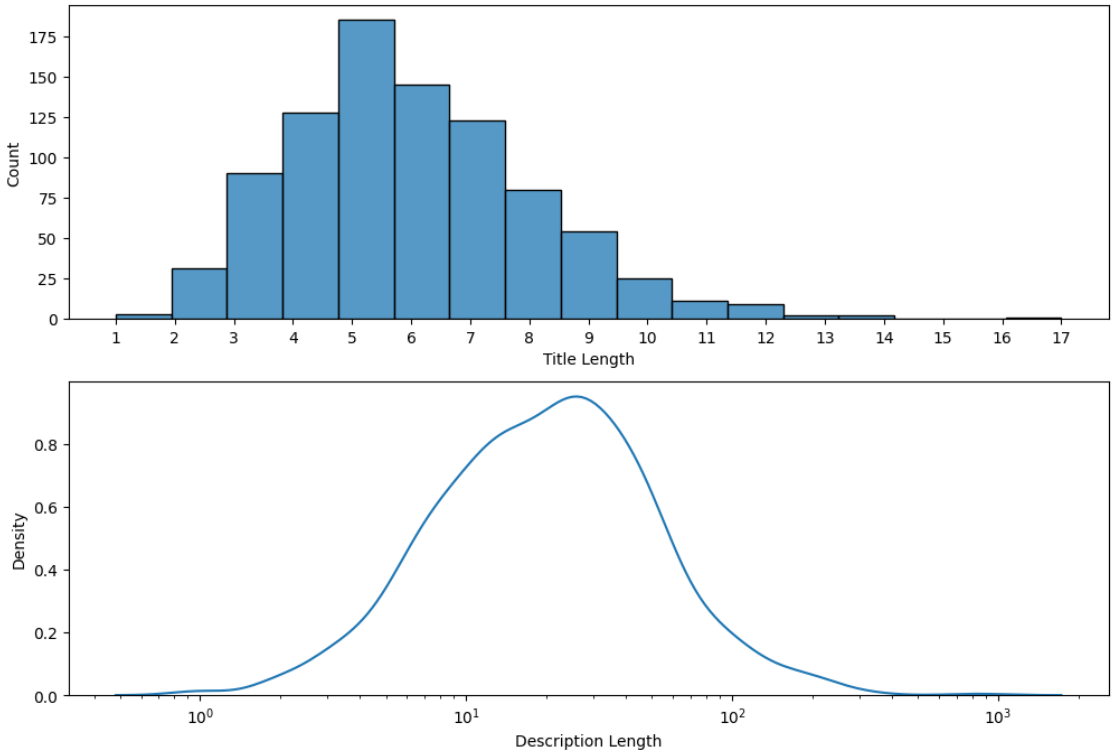
Plot the histogram of the title length and KDE of the description length (exclude 0):

[ ]:
```
plt.figure(figsize=(12, 8))

plt.subplot(2, 1, 1)
plt.xticks(np.arange(0, max(title_lengths) + 1, 1))
plt.xlabel('Title Length')
sns.histplot(title_lengths, bins=max(title_lengths))

plt.subplot(2, 1, 2)
plt.xlabel('Description Length')
plt.xscale('log')
sns.kdeplot(description_lengths[description_lengths > 0])
```

[ ]:
```
<Axes: xlabel='Description Length', ylabel='Density'>
```



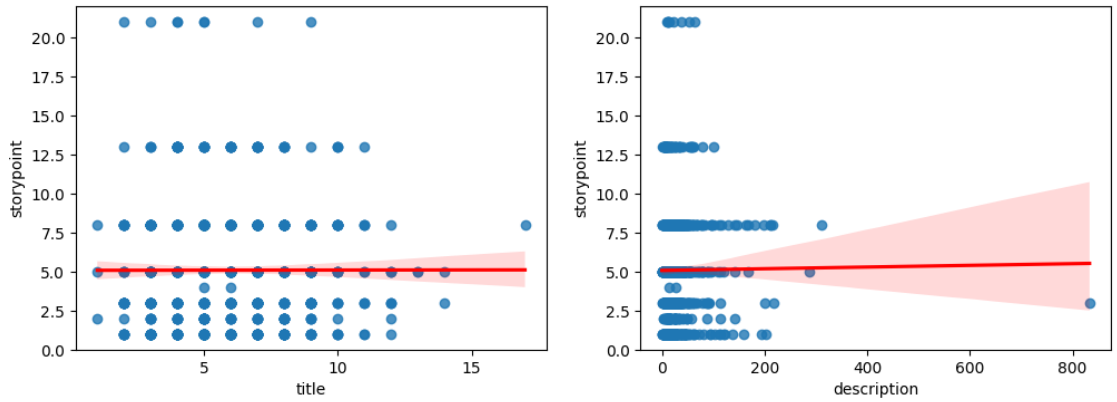
I think we should check the correlation between title length and description length:

[ ]:
```
plt.figure(figsize=(12, 4))

plt.subplot(1, 2, 1)
plt.xticks(np.arange(0, max(all_data['title'].apply(lambda x: len(x.split('␣
 ↪')))) + 1, 5))
sns.regplot(x=all_data['title'].apply(lambda x : len(x.split(' '))),
            y=all_data['storypoint'],
            line_kws={'color': 'red'})

plt.subplot(1, 2, 2)
sns.regplot(x=all_data['description'].apply(lambda x : len(x.split(' ')) if␣
 ↪type(x) != float else 0),
            y=all_data['storypoint'],
            line_kws={'color': 'red'})
```

[ ]:
```
<Axes: xlabel='description', ylabel='storypoint'>
```



Nope, no correlation at all

Let dive deeper in the input:

Title analysis:

[ ]:
```
count_vectorizer = CountVectorizer()
count_vectorizer.fit(all_data['title'])

dictionary = pd.DataFrame(list(count_vectorizer.vocabulary_.items()),␣
 ↪columns=['word', 'frequency'])
dictionary.sort_values(by='frequency', ascending=False, inplace=True)
print(dictionary.shape)
dictionary.head(10)
```

```
(1620, 2)
```

[ ]:

	word	frequency
543	zip	1619
713	xxe	1618
1349	xwwwformurlencoded	1617
67	xslttransformer	1616
711	xslt	1615
1584	xsdimport	1614
979	xsd	1613
336	xpath	1612
1374	xmltoxmlstreamreader	1611
1515	xmlstreamexception	1610

Description analysis:

[ ]:
```
count_vectorizer = CountVectorizer()
count_vectorizer.fit(all_data[all_data['description'].isnull() ==␣
 ↪False]['description'])
```

```
dictionary = pd.DataFrame(list(count_vectorizer.vocabulary_.items()),␣
 ↪columns=['word', 'frequency'])
dictionary.sort_values(by='frequency', ascending=False, inplace=True)
print(dictionary.shape)
dictionary.head(20)
```

```
(5265, 2)
```

[ ]:

	word	frequency
3326	zoombies	5264
1970	zipped	5263
4881	zipassistantnameother	5262
896	zip	5261
3051	zero	5260
5219	youve	5259
4757	youre	5258
242	youll	5257
2571	youd	5256
2123	yet	5255
1495	years	5254
4873	yearmarital	5253
2396	xyzconfig	5252
4080	xxxxxxxxxxxx	5251
4084	xxxxxxxxx	5250
3552	xxxxx	5249
778	xxxprocessingtimemonitor	5248
3003	xxxaware	5247
2464	xxe	5246
4835	xssequence	5245

Yet I don't find any thing special about the words in input except so many things are bad.

1.4.2 Solving strategies

My first intuitation in this problem is that the hard part is not on the algorithm we use, it is on the **embedding** part. Therefore, in case the given embedded datasets work not properly, I will use a better embedding method which is **Bidirectional Encoder Representations from Transformers (BERT)**. Also, I will try an old way to embedding the text too: **Bag of words**.

In conclusion, I will have 4 ways to embed the text: - doc2vec (already available) - Look up (already available) - Bag Of Words - BERT

About algorithm, I will try all the regression algorithm that may give a good result:

- Ridge Regressor
- Support Vector Regressor
- Random Forest Regressor
- Gradient Boosting
- XGBoost
- Lightgbm

- Blended

Maybe, we can change the problem to the classification problem with 100 labels (desparation confirmed). In the classification problem, I will use: - Support Vector Classifier - Softmax Regression (Multinomial Logistic Regression) - Random Forest - Adaboost - XGBoost

Thanks to the libaries, the implementation of all the algorithm shrinks to its minimum form.

At last, there is still a situation that all of mentioned model don't give a good result. This gamble is thrilling (hopeless).

"But would you lose?"

Nah, I'd win.